SYCL Hands-on Lab – Cambridge Version

Stephen Blair-Chappell
Intel oneAPI Certified Instructor
October 2024

intel

1. Getting Started

1. Log on to SWIRLES

2. Get list of intel tools via spack or module

```
spack list | grep intel
module spider oneapi
```

3.Load modules & environment

```
module load  intel-oneapi-compilers
```

4. Check versions of compiler

```
icpx --version
```

5. Check GPU Status (only on NVIDIA SYSTEMS)

```
nvidia-smi
```

6. List sycl accelerators available

```
sycl-ls
```

Hands-On Exercise 1 (5 minutes)

1) Use an editor to create code example(ex1.cpp)

2) Compile code

```
icpx -fsycl ex1.cpp -o ex1
```

3) Run program

```
./ex1
```

QUESTIONS:

- *Which lines of the code runs on the accelerator?*
- *Which Accelerator was used?*

```
#include <iostream>
#include <sycl/sycl.hpp>

int main() {
    try {
        // Step 1: Create a SYCL queue for the default accelerator
        sycl::queue q;

        // Print out the vendor and device name of the selected accelerator
        std::cout << "Running on: "
                  << q.get_device().get_info<sycl::info::device::vendor>()
                  << " "
                  << q.get_device().get_info<sycl::info::device::name>()
                  << std::endl;

        // Step 2: Offload "Hello World" using a lambda function in a kernel
        q.submit([&](sycl::handler &h) {
            // Create a SYCL stream object
            sycl::stream os(1024, 128, h);

            h.single_task([=]() {
                // Print "Hello World" from the accelerator using sycl::stream
                os << "Hello from SYCL!" << sycl::endl;
            });
        }).wait();
    } catch (sycl::exception const &e) {
        std::cout << "An exception occurred: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Hands-On Exercise 2(10 minutes)

1) Use an editor to create code example(ex2.cpp)

2) Compile code
`icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda ex2.cpp -o ex2`

3) Use the SYCL_PI_TRACE to confirm what accelerator the code ran on.

```
SYCL_PI_TRACE=1 ./ex2
```

QUESTIONS:

- *Which line of the code causes code to be associated with gpu?*
- *Can I change this behaviour at runtime with the* ONEAPI_DEVICE_SELECTOR ?

```
ONEAPI_DEVICE_SELECTOR="*:cpu" ./ex2
```

- Why did we use the options `-fsycl-targets=` ?

```
#include <iostream>
#include <sycl/sycl.hpp>

int main() {
    try {
        // Step 1: Create a SYCL queue for gpu accelerator
        sycl::queue q(sycl::gpu_selector_v);

        // Print out the vendor and device name of the selected accelerator
        std::cout << "Running on: "
                  << q.get_device().get_info<sycl::info::device::vendor>()
                  << " "
                  << q.get_device().get_info<sycl::info::device::name>()
                  << std::endl;

        // Step 2: Offload "Hello World" using a lambda function in a kernel
        q.submit([&](sycl::handler &h) {
            // Create a SYCL stream object
            sycl::stream os(1024, 128, h);

            h.single_task([=]() {
                // Print "Hello World" from the accelerator using sycl::stream
                os << "Hello from SYCL!" << sycl::endl;
            });
        }).wait();
    } catch (sycl::exception const &e) {
        std::cout << "An exception occurred: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Hands-On Exercise 3(10 minutes)

1) Use an editor to create code example(ex3.cpp)

2) Compile code

```
icpx -fsycl -fsycl-targets=nvptx64-nvidia-cuda
ex3.cpp -o ex3
```

3) Use the SYCL_PI_TRACE to confirm what accelerator the code ran on.

```
SYCL_PI_TRACE=1 ./ex3
```

QUESTIONS:

- *Which line of the code causes code to be associated with gpu?*
- *Can I change this behaviour at runtime with the* ONEAPI_DEVICE_SELECTOR ?

```
ONEAPI_DEVICE_SELECTOR="*:cpu" ./ex3
```

- What happens if you don't use the options `-fsycl-targets=` ?
- How might I make sure that code running on my laptop gives priority of NVIDIA GPU over Intel GPU

```
[opencl:cpu][opencl:0] Intel(R) OpenCL, 13th Gen Intel(R) Core(TM) i9-13900HX OpenCL 3.0 (Build
0) [2024.18.6.0.02_160000]
[opencl:gpu][opencl:1] Intel(R) OpenCL Graphics, Intel(R) Graphics [0xa788] OpenCL 3.0 NEO
[23.43.27642.52]
[level_zero:gpu][level_zero:0] Intel(R) Level-Zero, Intel(R) Graphics [0xa788] 1.3 [1.3.27642]~
[cuda:gpu][cuda:0] NVIDIA CUDA BACKEND, NVIDIA GeForce RTX 4090 Laptop GPU 8.9 [CUDA 12.5]
```

```cpp
#include <sycl/sycl.hpp>
#include <iostream>

// Define a custom device selection function
int custom_device_selector(const sycl::device& dev) {
    // Prioritize GPUs
    if (dev.is_gpu()) {
        return 1000;  // High score for GPUs
    }
    return -1;  // Ignore non-GPU devices
}

int main() {
    // Create a SYCL queue using the custom device selector function
    sycl::queue q(custom_device_selector);

    // Print the name of the selected device
    std::cout << "Running on: "
              << q.get_device().get_info<sycl::info::device::name>()
              << std::endl;

    return 0;
}
```

Hands-On Exercise 4 & 5 (20 minutes)

1) Use an editor to create code example(**ex4.cpp**) and fill in the missing code

2) Compile code
icpx -fsycl ex4.cpp -o ex4

3) Use the SYCL_PI_TRACE to confirm what memory allocations and memory transfers take place

SYCL_PI_TRACE=2 ./ex4

4) Save a copy of ex4.cpp as **ex5.cpp** and change the code so it uses EXPLICIT memory transfer rather than IMPLICIT memory transfer

5) Compile code
icpx -fsycl ex5.cpp

6) Use the SYCL_PI_TRACE to confirm what memory allocations and memory transfers take place

SYCL_PI_TRACE=2 ./ex5

QUESTIONS:

- *Explain the difference between the two code examples?*
- Is the trace different between the two examples – can you explain why?

```
#include <iostream>
#include <sycl/sycl.hpp>

int main() {
    // Step 1: Create a SYCL queue
    // TODO                              ;

    // Step 2: Allocate shared memory using USM
    const int N = 16;
    int *data =                              ;

    // Step 3: Initialize the array on the host
    for (int i = 0; i < N; i++) {
        data[i] = i;
    }

    // Step 4: Submit a kernel to the device to add 5 to each element
    q.parallel_for(N, [=](sycl::id<1> i) {
        /* Add 5 to each element of 'data' */
         //TODO                          ;
    }).wait();

    // Step 5: Print the updated values on the host
    for (int i = 0; i < N; i++) {
        std::cout << "data[" << i << "] = " << data[i] << std::endl;
    }

    // Step 6: Free the allocated memory
    // TODO                              ;

    return 0;
}
```

Hands-On Exercise 6(10 minutes)

1) Use an editor to create code example(ex6.cpp)

2) Compile code

```
icpx -fsycl ex6.cpp -o ex6
```

3) Use the SYCL_PI_TRACE to confirm what accelerator the code ran on.

```
SYCL_PI_TRACE=1 ./ex6
```

4)Create dependency graph and convert to png

```
SYCL_PRINT_EXECUTION_GRAPH=always ./ex6
dot -Tpng <filename> -o <outname>
```

This only works if graphviz is installed

5) Edit the code so the dependencies are implemented another way.

```
#include <sycl/sycl.hpp>
using namespace sycl;
constexpr int N = 42;

int main(int argc, char const *argv[])
{
    queue q;

    // Task A
    auto eA = q.submit([&](handler &h)
                       {
        sycl::stream out(1024, 256, h);
        h.parallel_for(N, [=](id<1>)
                       { out << "A"; }); });
    eA.wait();

    // Task B
    auto eB = q.submit([&](handler &h)
                       {
        sycl::stream out(1024, 256, h);
        h.parallel_for(N, [=](id<1>)
                       { out << "B"; }); });

    // Task C
    auto eC = q.submit([&](handler &h)
                       {
        sycl::stream out(1024, 256, h);
        h.depends_on(eB);
        h.parallel_for(N, [=](id<1>)
                       { out << "C"; }); });

    // Task D
    auto eD = q.submit([&](handler &h)
                       {
        sycl::stream out(1024, 256, h);
        h.depends_on({eB,eC});
        h.parallel_for(N, [=](id<1>)
                       { out << "D"; }); });

    return 0;
}
```

Hands-On Exercise 7 (30 minutes)

```
//==============================================================
// Copyright © Intel Corporation
//
// SPDX-License-Identifier: MIT
// =============================================================
#include <sycl/sycl.hpp>
using namespace sycl;

static const int N = 1024;
int main() {
  queue q;
  std::cout << "Device : " << q.get_device().get_info<info::device::name>() << "\n";

  //intialize 2 arrays on host
  int *data1 = static_cast<int *>(malloc(N * sizeof(int)));
  int *data2 = static_cast<int *>(malloc(N * sizeof(int)));
  for (int i = 0; i < N; i++) {
    data1[i] = 25;
    data2[i] = 49;
  }

  //# STEP 1 : Create USM device allocation for data1 and data2
  //# YOUR CODE GOES HERE

  //# STEP 2 : Copy data1 and data2 to USM device allocation
  //# YOUR CODE GOES HERE

  //# STEP 3 : Write kernel code to update data1 on device with square of its value
  q.parallel_for(N, [=](auto i) {

    //# YOUR CODE GOES HERE
  });

  //# STEP 3 : Write kernel code to update data2 on device with square of its value
  q.parallel_for(N, [=](auto i) {

    //# YOUR CODE GOES HERE
  });

  //# STEP 5 : Write kernel code to add data2 on device to data1

  q.parallel_for(N, [=](auto i) {

  //# YOUR CODE GOES HERE
  });

  //# STEP 6 : Copy data1 on device to host
  //# YOUR CODE GOES HERE

  //# verify results
  int fail = 0;
  for (int i = 0; i < N; i++) if(data1[i] != 12) {fail = 1; break;}
  if(fail == 1) std::cout << " FAIL"; else std::cout << " PASS";
  std::cout << "\n";

  //# STEP 7 : Free USM device allocations
  //# YOUR CODE GOES HERE

  //# STEP 8 : Add event based kernel dependency for the Steps 2 - 6

  return 0;
}
```

Other SYCL - LABs

- Request Jupyter session
- Clone the oneAPI-Samples
- (everything):

```
git clone https://github.com/oneapi-src/oneAPI-samples.git
```

OR

- Or use oneapi-cl and create new C++ Project based on:

```
oneAPI-samples/DirectProgramming/C++SYCL/Jupyter/oneapi-essentials-training
```

SYCLOMATIC -LAB

- Clone nvidia examples:

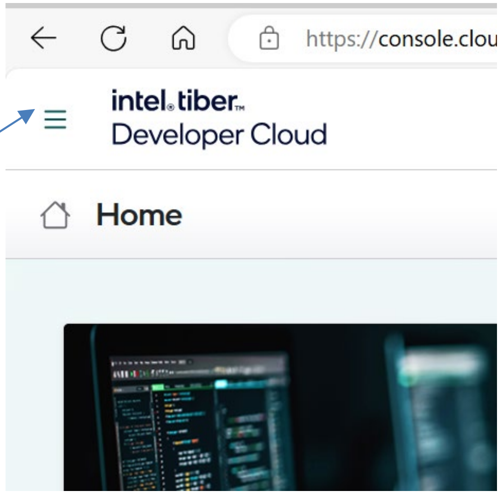
```
git clone https://github.com/NVIDIA/cuda-samples.git
```

- Choose one of the samples to migrate.

(Not all samples will migrate successfully!)

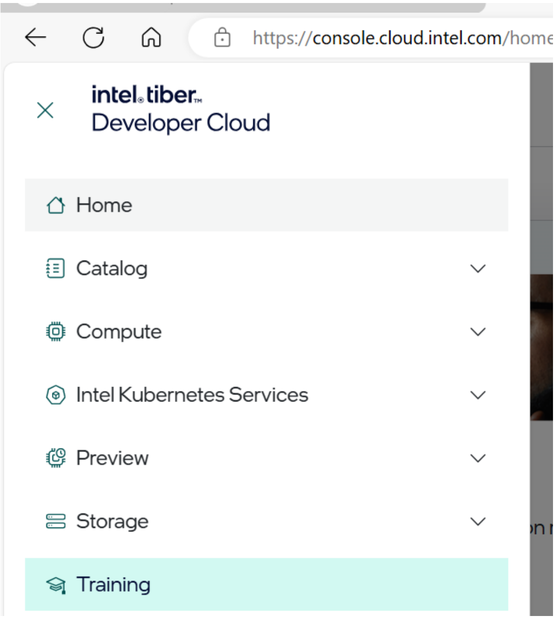
intel®

SYCL Programming Hands-on

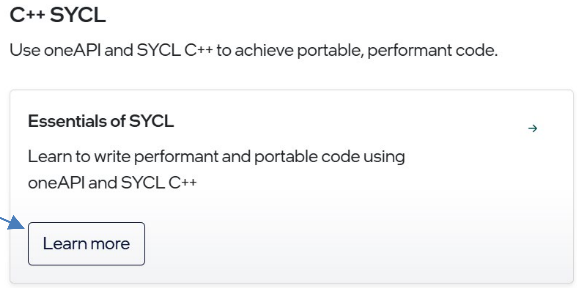


Log in to https://console.cloud.intel.com/

1. Click here to get the dropdown menu:
2. Choose training:



3. Scroll down and click 'Learn more in 'Essentials of SYCL C++'



4. Click 'Launch Jupyter Notebook